REMARKS/ARGUMENTS

This Amendment and the following remarks are intended to fully respond to the Office Action dated February 9, 2005. In that Office Action, claims 1-51 were examined, and all claims were rejected. More specifically, claims 1-6, 15-26, 31-32, and 34-51 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Jagannathan et al. (USPN 5,692,193) in view of Pinter et al. (USPN 6,457,023); and claims 7-14, 27-30, and 31 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Jagannathan et al. in view of Pinter et al. in view of Benayon et al. (USPN 5,809,554). Reconsideration of these rejections, as they might apply to the original and amended claims in view of these remarks, is respectfully requested.

In this Response, no claims have been amended; and no claims have been canceled.

Claim Rejections - 35 USC §103

Claims 1-6, 15-26, 31-32, and 34-51 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Jagannathan et al. (USPN 5,692,193) in view of Pinter et al. (USPN 6,457,023).

The examiner asserts that Jagannathan teaches the invention as claimed, including analysis during code compilation to distinguish between thread specific data and shared data and configuring the target program to allocate thread specific data to a thread specific heap and shared data to the shared heap.

It is respectfully submitted that there is a fundamental difference between the claimed "thread-specific heaps" in Applicant's application and the "local heaps" disclosed in the Jagannathan patent. They are not the same and do not operate in the same way. The objects placed in Applicant's thread-specific heaps are **guaranteed** to be local to a thread whereas the objects placed in Jagannathan's local heaps are **presumed** to be local to a thread. There is simply no suggestion of such "front end" verification of local thread heap objects in Jagannathan et al.

In Jagannathan's system, if the presumption turns out to be wrong and therefore a reference to an object in a local heap somehow escapes, the object (and other objects transitively reachable from the object) must be copied into a global heap.

Jagannathan states, at column 21, lines 44-52:

This is because any object that is referenced from a shared object is also a shared object and, therefore must reside in a shared heap. This constraint on shared heaps is enforced by ensuring that references stored in shared heaps refer to objects that are (a) either in a shared heap, or (b) <u>allocated in a local heap and garbage collected into a shared one.</u> (Emphasis added) That is, the graphs of objects reachable from the referenced object must be copied into or located in the shared heap. The overheads of this memory model depend on how frequently references to objects allocated on local heaps escape.

This operational system is based on a presumption. That is, that in Jagannathan's system, the objects placed in local heaps are PRESUMED to be local to a thread. The runtime checks for violations of the assumption of thread locality of objects have a cost; most likely the check is incorporated into a write barrier, and the garbage collector must be designed to work with the on-demand copying out of objects from a local heap into a global heap.

In contrast, in Applicant's claimed system, the program analysis performs a proof that the objects in the thread-specific heaps are never accessed by threads other than the one owning the thread-specific heap in which each object resides. Applicant's specification, pages 7, line 14 to page 8, line 18 states:

> Thread-specific data is distinguished from shared data in that **thread-specific data is determined to be unreachable from outside of a given program thread.** An exemplary method of determining whether program data is deemed potentially reachable from outside of a given program thread is called "thread escape analysis", although other analysis methods are also contemplated within the scope of the present invention. **Program data that is not determined to be "thread-specific data" is deemed "shared data".** As such, shared data includes program data that is deemed potentially reachable from multiple program threads. Furthermore, a thread-specific heap may exist for each program thread of a target program, and program data specific to each program thread are allocated appropriately to the associated thread-specific heap.
>
> During runtime, thread-specific data is allocated to thread-specific heaps, and shared data is allocated to one or more shared heaps. At some appropriate time during execution, a garbage collector module reclaims memory for unneeded program data from one or more of the target program's heaps. **Because no thread-specific data within a thread-specific heap is**

14

determined to be reachable from outside the associated program thread, the thread-specific heap can be collected without impacting (e.g., suspending or synchronizing with) the execution of other program threads in the target program. In addition, the impact of garbage collection of the shared heap on program threads can be minimized because the program threads can substantially continue their execution so long as they do not access program data in the shared heap.

It should be understood that exemplary program analysis that determines "reachability" at compile time is based on a conservative estimate of what may occur at runtime. Therefore, in an embodiment of the present invention, the **program analysis at compile time proves that an object is "thread-specific" by proving that the object will never be referenced by multiple threads. However, an object may be deemed potentially reachable by multiple threads (i.e., "shared"), if it cannot be proven that the object will never be referenced by multiple threads (i.e., even though it has not been proven that the object will indeed be referenced by multiple threads during execution). In contrast, during runtime, reachability can be specifically determined, thereby making the conservative approximation unnecessary during runtime.** (emphasis added)

, and page 11, lines 17-19:

An object that is proven to be reachable by only a single program thread is referred to as a "thread-specific object". In contrast, if the object is deemed potentially reachable by more than one program thread, the object is referred to as a "shared object".

Therefore, in Applicant's invention as claimed, there is **no need** for any checks of violations of an assumption of thread locality of objects. In addition, the garbage collector algorithms for the thread-specific heaps do not need to allow for on-demand copying out of objects from a thread-specific heap into a global heap as in Jagannathan. This differs substantially from Jagannathan.

Jagannathan suggests using program analysis **during runtime** to determine which objects **are likely** to escape and should therefore initially be allocated in a shared heap rather than initially being allocated in a local heap and then subsequently copied out into a shared heap. In Applicant's system, program analysis is used **at compilation time** to **prove** that certain objects are thread-specific such that runtime analysis is NOT called for, as "escape" is not possible. In contrast, Jagannathan makes a presumption and does not prove thread specificity.

15

Independent claims 1, 25, 34, 35, 40 and 47 each recite "**proven** thread-specific data" as defined precisely in Applicant's specification. Therefore it is respectfully submitted that Jagannathan et al does not disclose or teach Applicant's CLAIMED invention as defined in these independent claims.

Claims 7-14, 27-30, and 31 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Jagannathan et al. in view of Pinter et al. in view of Benayon et al. (USPN 5,809,554).

The examiner asserts that Pinter "teaches the invention as claimed including analyzing the target program during code compilation (col. 1, line 65-col. 2, line 6) to identify proven thread-specific data (col. 4, lines 36-43; col. 5, lines 5-18; col. 8, lines 42-49; col. 9, lines 21-31)."

Pinter clearly does not address the deficiencies in Jagannathan as pointed out above.

First, it is respectfully submitted that the examiner is mistaken in asserting that Pinter et al teaches analyzing the target program during code compilation. Col. 1, lines 65-col. 2, line 6 state:

> In other approaches attempts have been made to increase the efficiency of data flow analysis. Data flow analysis computes information about the potential behavior of program in terms of the definitions and uses of data objects. Such data flow information is important for optimizing compilers, program environments, and understanding tools. It can also be used in a software-testing system or to provide compiler and runtime support for the parallel execution of programs originally written in sequential languages.

There is nothing in this passage that teaches analyzing a target program <u>during</u> target code compilation.

Second, Pinter is directed to an entirely different problem. Pinter teaches organizing garbage collection around heap lifetimes. The examiner asserts that Pinter teaches identifying **proven** thread-specific data, and references Pinter et al text portions in support. These references do not teach identification of proven thread-specific data. These references teach identifying objects and their caller-callee relationships in the program (col. 4, lines 36-43).

Dependent claims 7-14, 27-30, and 33 stand rejected as obvious over Jagannathan in view of Pinter et al and further in view of Benayon et al. The examiner agrees that Benayon does not specifically address identifying thread specific objects and allocation to thread specific heaps during compilation, but asserts that this step is taught by Jagannathan et al. As discussed above, the analysis which proves that an object is thread specific is not performed in Jagannathan et al. A presumption is merely made. Since the rejected claims depend from Applicant's independent claims 1, 25, 34, 35, and 40, and Pinter et al does not fill the deficiencies noted above, and Benayon et al does not disclose **proven thread-specific data** as claimed, the rejection of the dependent claims 7-14, 27-30, 33 should be withdrawn. These claims are clearly believed to be patentable for the reasons set forth above.

Conclusion

It is believed that no further fees are due with this Response. However, the Commissioner is hereby authorized to charge any deficiencies or credit any overpayment with respect to this patent application to deposit account number 13-2725.

In light of the above remarks and amendments, it is believed that the application is now in condition for allowance and such action is respectfully requested. Should any additional issues need to be resolved, the Examiner is requested to telephone the undersigned to attempt to resolve those issues.
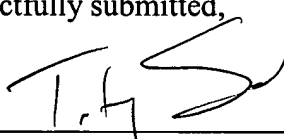
Respectfully submitted,

5/9/05

Date

27488
PATENT TRADEMARK OFFICE

Timothy B. Scull, Reg. No. 42,137
Merchant & Gould P.C.
P.O. Box 2903
Minneapolis, MN 55402-0903
(303) 357-1648
(303) 357-1671 (fax)

17